CS103L SPRING 2020

UNIT 7: FILE I/O

TEXT

I/O STREAMS IN C++

- I/O: short for input/ouput
- Older term from mainframe days for getting data into and out of your program
- C++ offers object oriented abstractions to make using I/O easier
- We use the “stream” abstractions through cin, cout, ifstreams, ofstreams

STREAM ABSTRACTION

- The stream objects we use model I/O as a “stream”
- A stream is a series of data (usually single characters) one after the other
  - Narrow but long, like a stream
- We interact with streams differently depending on if they are input or output streams
  - Input streams, we extract characters from the stream into our program
  - Output streams, we insert characters from our program into the stream

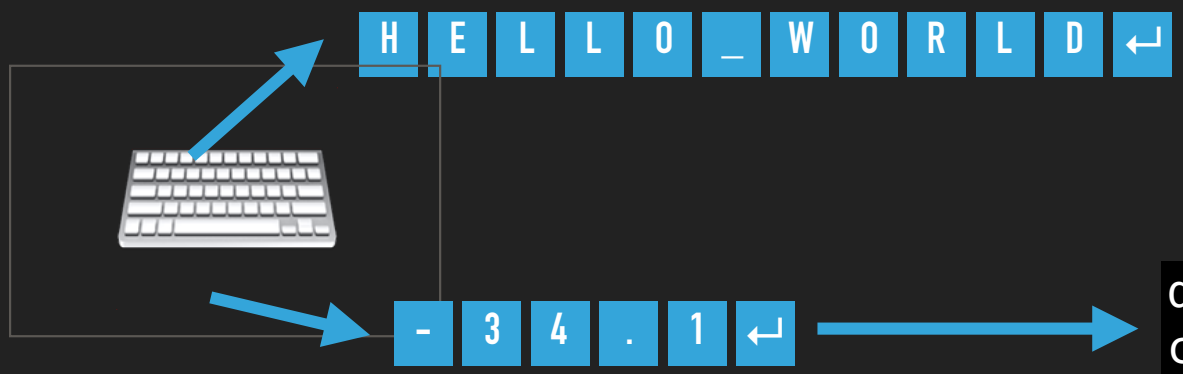
INPUT STREAMS

- Input streams are a series of characters available for our program to read
- For now, just “cin” where the characters come for keyboard
- Next, file input streams
- Eventually internal input streams (stringstreams)
- Also, network streams

CIN INPUT STREAM

- cin keeps a buffer of characters read from the keyboard
- Allows us to read them one-by-one
- Or formatted - powerful abstraction!

```
char c;
cin >> c;
```




```
double x;
cin >> x;
```

TEXT

COUT OUTPUT STREAM

- Using cout with the << operator takes data from our program and inserts it into an output stream
  - The stream is usually shown on the terminal
- Output is formatted: cout decides how to present the data
  - int or double: as a number
  - char: as a single character
  - char*: as a string (follow the pointer, print the chars until NULL
  - Any other pointer type: memory address as hexadecimal

TEXT

I/O STREAM ABSTRACTION

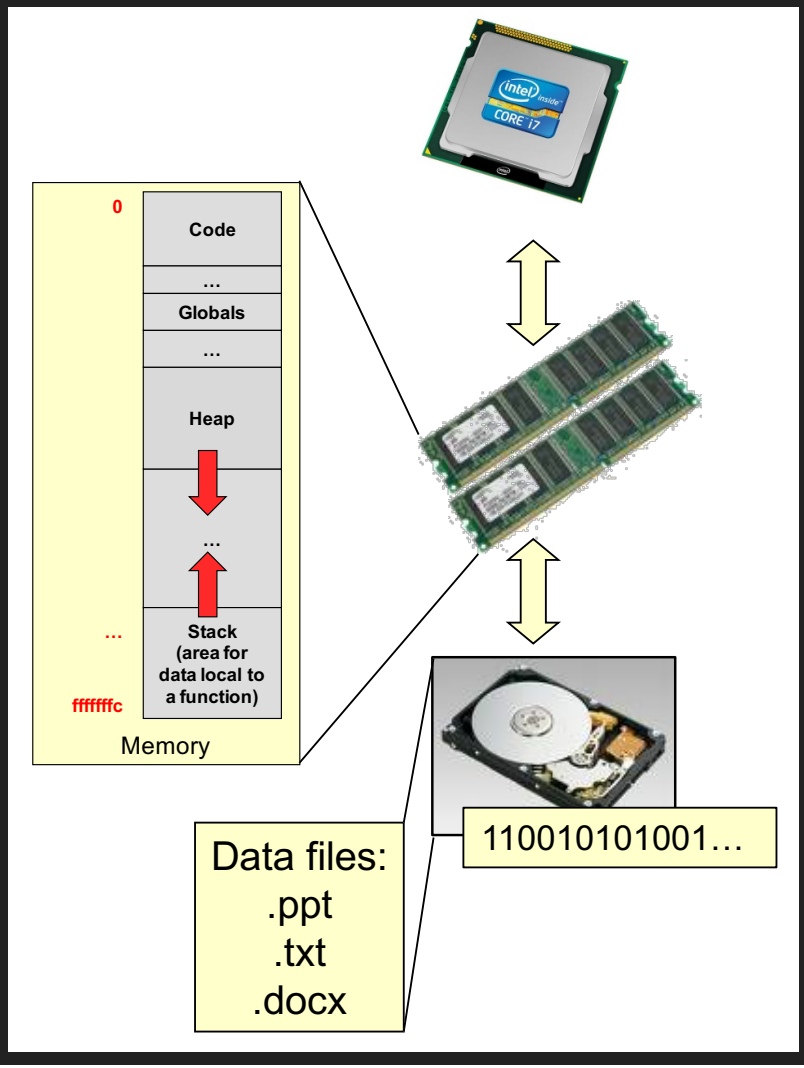
- Why is the I/O stream abstraction powerful?
- Once you can use one type of I/O stream, you can use any!
- They all define the same interface

FILE I/O

- How can we get data into our programs?
  - cin
  - Command line arguments
- How can we get data out?
  - Just cout for now
- Now we add file I/O through file streams

NOTE ON COMPUTER ARCHITECTURE

- CPU can talk to RAM and I/O devices
  - Like keyboard, terminal, disk controllers
- Our program can only interact with data in RAM, specifically *its* memory space only
- OS and C++ provide interfaces and abstractions so that C++ programs can do I/O (keyboard, terminal, disk, network)



TEXT

IMPORTANT NOTE!

- Remember: everything we do in C++ is an interaction with a variable or memory address
- In order to work with data in C++ we must put it in a variable
  - After the data is loaded in a variable (or memory) we can process it
- Already seen with cin/cout, now adding file I/O
- Everything we're learning today is how to get data out of a file into a variable (and the other way around)

TEXT

FILE I/O INTERFACE

- #include <fstream>
- ifstream object type
  - Lets us open a file, extract data, close file
- ofstream object type
  - Lets us open a file for writing, insert data, save and close the file

BINARY VS. TEXT DATA

- Granularity of data in file world is the byte
  - Files are modeled as streams of bytes, one-after-another
- Two types:
  - Text files: bytes are interpreted as a series of ASCII character
    - .txt file, code, csv file, HTML
  - Binary files: bytes are interpreted as binary data as specified by the file format:
    - Image files, videos, audio, compiled executables
- Example: store integer 103 in a file
  - Text file: '1' '0' '3' - takes up 3 bytes
  - Binary: could store as single byte 0x67 or if an 32-bit int, 0x00000067 - depends on file format

TEXT FILE I/O

- We only do text file I/O in 103
- #include <fstream>
- Use ifstream object to read from a file
  - Works *exactly* like cin, use >> operator
- Use ofstream object to write to a file
  - Works *exactly* like cout use << operator

input.txt

```
5 -3.5
```

output.txt

```
Int from file is 5
Double from file is -3.5
```

```
#include <iostream>
#include <fstream>
using namespace std;

int main ()

{

  int x; double y;

  ifstream ifile ("input.txt");

  if(  ifile.fail() ){  // able to open file?
    cout << "Couldn't open file" << endl;
    return 1;
  }

  ifile >> x >> y;
  if (  ifile.fail() ){
    cout << "Didn't enter an int and double";
    return 1;
  }

  ofstream ofile("output.txt");

  ofile << "Int from file is " << x << endl;
  ofile << "Double from file is " << y << endl;

  ifile.close();
  ofile.close();

  return 0;
 }
```

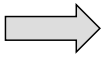
GETTING LINES OF TEXT

- >> skips then stops at whitespace
- Sometimes we want a whole *line* of text
- stream.getline(char *buf, int bufsize)
  - cin.getline(buf, 100); ifile.getline(buf, 100);
- Program reads whole lines of text, prints out

```
#include <iostream>
#include <fstream>
using namespace std;

int main ()

{

  char myline[100]; int i = 1;

  ifstream ifile ("input.txt");

  if(  ifile.fail() ){  // can't open?
    return 1;
  }

  ifile.getline(myline, 100);
  while ( ! ifile.fail()) {
   cout << i++ << ": " << myline << endl;
   ifile.getline(myline, 100);
  }

  ifile.close();
  return 0;
}
```

input.txt

```
The fox jumped over the log.\n
The bear ate some honey.\n
The CS student solved a hard problem.\n
```



```
1: The fox jumped over the log.
2: The bear ate some honey.
3: The CS student solved a hard problem.
```

IN-CLASS EXERCISES

- wget http://ee.usc.edu/~redekopp/cs103/file_io_ex.tar

- sum_from_file
- reverse_it
- countnum

OTHER INPUT STREAM FEATURES

- Getting lines of text
  - Seen already with files, applies to cin as well
  - >> operator skips, then stops at whitespace
- Use stream.getline(char* buf, int max) to read a line of text including whitespace, but not the '\n')
- .getline() reads up to max-1 characters and inserts the NULL-char for you

```
#include <iostream>
using namespace std;

int main ()
{
  char mytext[80];
  cout << "Enter your full name" << endl;
  cin.getline(mytext, 80);

  int last=0;
  for(int i=0; i<80; i++){
    if(mytext[i] == ' '){
       last = i;
       break;
    }
  }
  cout << "Last name starts at index: ";
  cout << last << endl;
  return 0;
 }
```

CHECKING INPUT STREAM ERRORS

- Until now we didn't know how to detect errors with cin (or ifstreams)
- Input streams have .fail() method - returns 'true' if something has gone wrong
  - Couldn't open file, asked for int but got 'abc', reached end of input file, etc.
- Once an error occurs .fail() returns 'true' until .clear() is called.

```
#include <iostream>
using namespace std;

int main ()

{

  int x;
  cout << "Enter an int: " << endl;
  cin >> x;  // What if the user enters:
             //    "abc"


  // Check if we successfully read an int
  if(  cin.fail() ) {
    cout << "Error:  I said enter an int!";
    cout << " Now I must exit!" << endl;
    return 1;
  }

  cout << "You did it!  You entered an int";
  cout << " with value: " << x;

  return 0;
 }
```

EXAMPLE

- wget http://ee.usc.edu/~redekopp/cs103/cinfail.cpp
- cinfail example

UNDERSTANDING INPUT STREAMS AND .FAIL()

- User enters value “512” at 1st prompt, enters “123” at 2nd prompt

Code	Variable	cin
int x=0;	X = 0	cin =
cout << “Enter X: “;	X = 0	cin = 5 1 2 \n
cin >> x;	X = 512	cin = \n
	cin.fail() is false	
int y = 0;	Y = 0	cin = \n
cout << “Enter Y: “;	Y = 0	cin = \n 1 2 3 \n
cin >> y;	Y = 123	cin = \n
	cin.fail() is false	

UNDERSTANDING INPUT STREAMS AND .FAIL()

. User enters value “23abc” at 1st prompt, 2nd prompt fails

```
int x=0;

cout << “Enter X: “;

cin >> x;

int y = 0;

cout << “Enter Y: “;

cin >> y;
```

X = 0 cin =

X = 0 cin = 2 3 a b c \n

X = 23 cin = a b c \n

cin.fail() is false

Y = 0 cin = a b c \n

Y = 0 cin = a b c \n

Y = xx cin = a b c \n

cin.fail() is true

UNDERSTANDING INPUT STREAMS AND .FAIL()

- User enters value "23 99" at 1st prompt, 2nd prompt skipped

int x=0;

X = 0 cin =

cout << "Enter X: ";

X = 0 cin = 2 3 ␣ 9 9 \n

cin >> x;

X = 23 cin = ␣ 9 9 \n

cin.fail() is false

int y = 0;

Y = 0 cin = ␣ 9 9 \n

cout << "Enter Y: ";

Y = 0 cin = ␣ 9 9 \n

cin >> y;

Y = 99 cin = \n

cin.fail() is false

UNDERSTANDING INPUT STREAMS AND .FAIL()

. User enters value “23 99” at 1st prompt, everything read as string

char x[80];

cout << “Enter X: “;

cin.getline(x, 80);

X = cin =

X = cin = 2 3 9 9 \n

X = 23 99 cin =

cin.fail() is
false

NOTE: \n character is discarded!

USING .FAIL, .IGNORE, .CLEAR

- You use .fail() to detect errors when reading data.
- .ignore(int n, char delim) clears out the buffer up to the next delim character, or a maximum of n
- .clear() resets .fail() flag, otherwise .fail() will continue to return ‘true’

```
#include <iostream>
using namespace std;

int main ()

{

  int x;
  cout << "Enter an int: " << endl;
  cin >> x;  // What if the user enters:
             //    “abc”


  // Check if we successfully read an int
  while(  cin.fail() ) {
    cin.clear(); // turn off fail flag
    cin.ignore(256, '\n'); // clear inputs
    cout << "I said enter an int: ";
    cin >> x;

  }

  cout << "You did it!  You entered an int";
  cout << " with value: " << x;

  return 1;
 }
```

TEXT

OS REDIRECTION AND PIPES

- Redirection and pipes give a secondary way to do file I/O with programs
- They are OS level services
- Performed on the command line with:
  - ‘<‘ input redirection
  - ‘>’ output redirection
  - | pipe output to input

INPUT REDIRECTION

- '<' input redirection
  - Takes the contents of a file and makes this the 'stdin' of your program
  - Extracting from cin with >> gets data from the file, not the keyboard
  - ./zombies < test1.txt

TEXT

OUTPUT REDIRECTION

- '>' takes the stdout (aka would normally go to terminal from cout) and redirects it to a file
- ./randgen 10 100 > random.txt

TEXT

PIPES

- Using the | (pipe) connects the stdout of one program to the stdin of another
- Everything you ‘cout’ in one program becomes the source for ‘cin’ in the 2nd
- ./randgen 10 100 | ./average
- wget http://ee.usc.edu/~redekopp/cs103/redir_pipe.tar

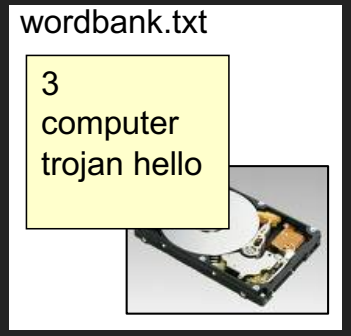
TEXT

REDIRECTION AND PIPING EXAMPLES

- http://ee.usc.edu/~redekopp/cs103/redir_pipe.tar
- randgen
- average

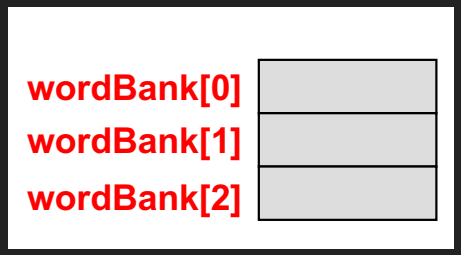
LAB 7 OVERVIEW

- Lab 7: re-do word scramble game to read the word bank words from a file.
- File format:
  - First line: number of words (N)
  - N words separated by white space



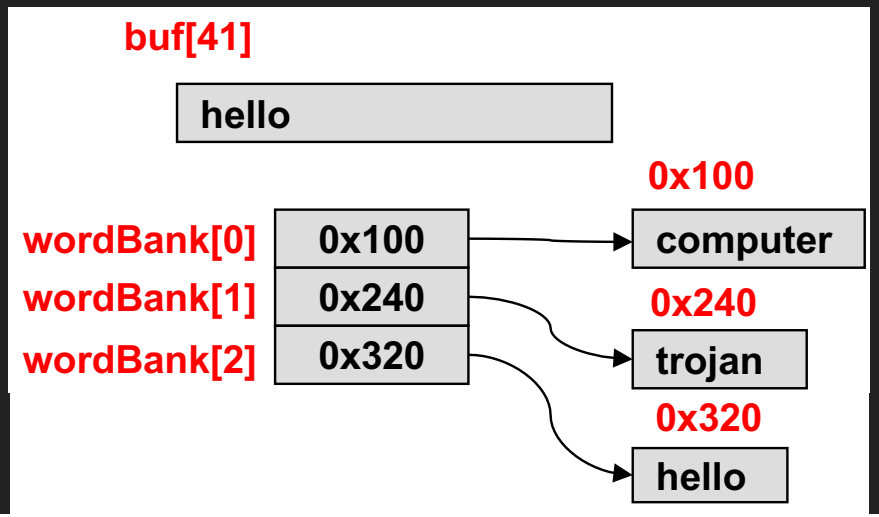
LAB 7 OVERVIEW

- Attempt to open the file. If that succeeds (ifile.fail() == false) go on
- Read in number of words to a variable
- If that succeeds (ifile.fail() == false), allocate (with dynamic memory) a word back array. wordBank will hold char*, so it has type char**



LAB 7 OVERVIEW

- In a loop (N times) read in a word to a temporary buffer and then allocate some memory to hold that word.
- Why two steps? (read then allocate?)
- Then copy word from temporary buffer to allocated memory



LAB 7 OVERVIEW

- Now you have a dynamically allocated wordBank[] that could hold an arbitrary number of words